(1)

WRDC-TR-90-8007
Volume III
Part 1

WRIGHT RESEARCH & DEVELOPMENT CENTER

# INTEGRATED INFORMATION SUPPORT SYSTEM (IISS)
Volume III - Configuration Management
Part 1 - Quality Assurance Plan

M. Foster

Control Data Corporation
Integration Technology Services
2970 Presidential Drive
Fairborn, OH 45324-6209

September 1990

Final Report for Period 1 April 1987 - 31 December 19[illegible]

MANUFACTURING TECHNOLOGY DIRECTORATE
WRIGHT RESEARCH AND DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 4543[illegible]

[illegible]
DAVID L. JUDSON, Project Manager
WRDC/MTI
Wright-Patterson AFB, OH 45433-6533

25 July 91
DATE

FOR THE COMMANDER:

[illegible]
BRUCE A. RASMUSSEN, Chief
WRDC/MTI
Wright-Patterson AFB, OH 45433-6533

25 July 91
DATE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

# REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION<br>Unclassified | 1b. RESTRICTIVE MARKINGS |
| 2a. SECURITY CLASSIFICATION AUTHORITY<br>2b. DECLASSIFICATION/DOWNGRADING SCHEDULE | 3. DISTRIBUTION/AVAILABILITY OF REPORT<br>Approved for Public Release;<br>Distribution is Unlimited. |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S)<br>QAP620320000 | 5. MONITORING ORGANIZATION REPORT NUMBER(S)<br>WRDC-TR-90-8007 Vol. III, Part 1 |
| 6a. NAME OF PERFORMING ORGANIZATION<br>Control Data Corporation;<br>Integration Technology Services<br>6b. OFFICE SYMBOL *(if applicable)* | 7a. NAME OF MONITORING ORGANIZATION<br>WRDC/MTI |
| 6c. ADDRESS *(City,State, and ZIP Code)*<br>2970 Presidential Drive<br>Fairborn, OH 45324-6209 | 7b. ADDRESS *(City, State, and ZIP Code)*<br>WPAFB, OH 45433-6533 |
| 8a. NAME OF FUNDING/SPONSORING ORGANIZATION<br>Wright Research and Development Center,<br>Air Force Systems Command, USAF<br>8b. OFFICE SYMBOL *(if applicable)*<br>WRDC/MTI | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUM.<br>F33600-87-C-0464 |
| 8c. ADDRESS *(City, State, and ZIP Code)*<br>Wright-Patterson AFB, Ohio 45433-6533 | 10. SOURCE OF FUNDING NOS.<br>PROGRAM ELEMENT NO.: 78011F<br>PROJECT NO.: 595600<br>TASK NO.: F95600<br>WORK UNIT NO.: 20950607 |
| 11. TITLE *(Include Security Classification)*<br>See Block 19 | |
| 12. PERSONAL AUTHOR(S)<br>Control Data Corporation: Foster, M. | |

| 13a. TYPE OF REPORT | 13b. TIME COVERED | 14. DATE OF REPORT *(Yr.,Mo.,Day)* | 15. PAGE COUNT |
|---|---|---|---|
| Final Report | 4/1/87-12/31/90 | 1990 September 30 | 20 |

16. SUPPLEMENTARY NOTATION

WRDC/MTI Project Priority 6203

17. COSATI CODES

| FIELD | GROUP | SUB GR. |
|---|---|---|
| 1308 | 0905 | |
| | | |

18. SUBJECT TERMS *(Continue on reverse if necessary and identify block no.)*

19. ABSTRACT *(Continue on reverse if necessary and identify block number)*

This document establishes the quality assurance standards and procedures to be applied for IISS software and documentation development, update, and test through 30 September 1990.

Block 11 - INTEGRATED INFORMATION SUPPORT SYSTEM (IISS)
Vol III - Configuration Management
Part 1 - Quality Assurance Plan

| | |
|---|---|
| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT<br>UNCLASSIFIED/UNLIMITED x SAME AS RPT. DTIC USERS | 21. ABSTRACT SECURITY CLASSIFICATION<br>Unclassified |
| 22a. NAME OF RESPONSIBLE INDIVIDUAL<br>David L. Judson | 22b. TELEPHONE NO. *(Include Area Code)*<br>(513) 255-7371<br>22c. OFFICE SYMBOL<br>WRDC/MTI |

DD FORM 1473, 83 APR

EDITION OF 1 JAN 73 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

QAP620320000
30 September 1990

## FOREWORD

This technical report covers work performed under Air Force Contract F33600-87-C-0464, DAPro Project. This contract is sponsored by the Manufacturing Technology Directorate, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio. It was administered under the technical direction of Mr. Bruce A. Rasmussen, Branch Chief, Integration Technology Division, Manufacturing Technology Directorate, through Mr. David L. Judson, Project Manager. The Prime Contractor was Integration Technology Services, Software Programs Division, of the Control Data Corporation, Dayton, Ohio, under the direction of Mr. W. A. Osborne. The DAPro Project Manager for Control Data Corporation was Mr. Jimmy P. Maxwell.

The DAPro project was created to continue the development, test, and demonstration of the Integrated Information Support System (IISS). The IISS technology work comprises enhancements to IISS software and the establishment and operation of IISS test bed hardware and communications for developers and users.

The following list names the Control Data Corporation subcontractors and their contributing activities:

| SUBCONTRACTOR | ROLE |
|---|---|
| Control Data Corporation | Responsible for the overall Common Data Model design development and implementation, IISS integration and test, and technology transfer of IISS. |
| D. Appleton Company | Responsible for providing software information services for the Common Data Model and IDEF1X integration methodology. |
| ONTEK | Responsible for defining and testing a representative integrated system base in Artificial Intelligence techniques to establish fitness for use. |
| Simpact Corporation | Responsible for Communication development |
| Structural Dynamics Research Corporation | Responsible for User Interfaces, Virtual Terminal Interface,and Network Transaction Manager design, development, implementation, and support. |

| | |
|---|---|
| Arizona State University | Responsible for test bed operations and support. |

QAP620320000
30 September 1990

TABLE OF CONTENTS

## LIST OF ILLUSTRATIONS

# SECTION 1

# INTRODUCTION

## 1.1 Quality Assurance

Any project large or small throughout its life cycle requires standards and quidelines to assure production and release compatibility, the efficient coordination of individual efforts, and the verification of correct performance. The application of these standards and guidelines are imperative for the Integrated Information Support System (IISS) software project with its composition of multi-tasked and multi-layered subsystems and an application that spans heterogeneous computers and heterogeneous system environments.

## 1.2 Quality Assurance Planning

The purpose of this document is to examine in a general way the major areas of concern that are a part of any quality assurance plan and to refer to the specific documents that provide the quidelines and conventions that were developed for IISS during the DAPro Project 6203. All specific quality assurance guidelines were put into specific user manuals to facilitate their distribution and use by developers and IISS administrators.

1. Software quidelines
2. Software configuration management
3. Documentation quidelines
4. Administrative quidelines
5. Testing quidelines
6. Life cycle activities

## SECTION 2

## ASPECTS OF QUALITY ASSURANCE

### 2.1 Scope

Quality assurance provisions should be defined in terms of software engineering standards, tools, methodologies, procedures, guidelines and control mechanisms, which in turn define the software development process. The following discussions involve software quality, software configuration management, life cycle activities, and system testing.

### 2.2 Approach to Software Quality Assurance

Some of the information in this section has been adapted from Software Quality and Assurance, A Survey, by Thomas J. McCabe & Associates.

#### 2.2.1 Software Quality and Quality Factors

The purpose of establishing formal software engineering standards, procedures and control mechanisms is to assure the production of high quality software. The term "software quality" involves a hierarchy of definitions as shown in Figure 2-1.

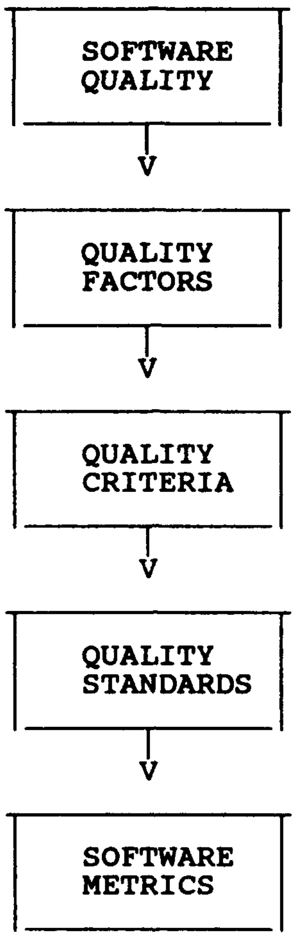

Figure 2-1. Hierarchy of Software Quality Concepts.

Software quality is first defined in terms of quality factors. Simply stated, quality factors are desirable characteristics or attributes associated with a software product. The restated purpose of establishinng a quality assurance program, therefore, is to instill the likelihood that important quality factors characterize all the software products software engineering methodologies, tools, procedures, standards and guidelines for ensuring that important quality factors are present within all software products.

These are quality factor examples:

1. Maintainability
2. Flexibility
3. Reliability
4. Portability
5. Testability

6. Reusability
7. Efficiency
8. Correctness

Informal definitions for these quality factors are shown in Figure 2-2.

| QUALITY FACTOR | | DEFINITION |
|---|---|---|
| MAINTAINABILITY | ---- | HOW EASILY CAN I FIX IT? |
| FLEXIBILITY | ---- | HOW EASILY CAN I CHANGE IT? |
| PORTABILITY | ---- | HOW EASILY CAN I REHOST IT? |
| RELIABILITY | ---- | ACCURATE ALL THE TIME? |
| TESTABILITY | ---- | HOW EASILY CAN I TEST IT? |
| REUSABILITY | ---- | CAN I REUSE IT? |
| EFFICIENCY | ---- | DOES IT OPTIMIZE RESOURCES (CPU, MEMORY)? |
| CORRECTNESS | ---- | DOES IT SATISFY THE SPECS? |
| INTEGRITY | ---- | IS IT SECURE? |

Figure 2-2. Definitions of Quality Factors.

The establishment of a formal software quality assurance program in any software development project should begin, therefore, with an identification of the important quality factors which contribute to and, in a sense, define what constitutes quality software for that project.

Quality factors also can be arranged according to their relative importance to a specific development effort. Maintainability and flexibility, for example, may be important to a systems engineering organization interested in marketing proprietary software packages and tailoring them to a number of customers each having special site-specific requirements. These factors, however, pale in importance to correctness and reliability when a software product on which lives will depend (e.g., space shuttle software) is being developed.

The meaning of software quality is not static. It varies depending on the goals of the development organization or the nature of each specific application or project.

### 2.2.2 Quality Criteria

Quality factors are in turn defined in terms of quality criteria which further define and help to establish the presence of the factor in a software product. Simplicity, modularity and consistency, for example, are importanat quality criteria for insuring a high degree of maintainabilty.

Quality criteria are used for determining or judging whether or not quality factors important to an organization or product are actually present. A quality criterion, therefore, is not an end unto itself. Itinstead can be thought of as a means to an end. Modularity, therefore, is one of possibly several means by which a high degree of maintainability is achieved.

Simple definitions for specific quality criteria are included in Figure 2-3. Quality factors and examples of associated quality criteria are illustrated in Figure 2-4.

| QUALITY CRITERIA | - | DEFINITIONS |
|---|---|---|
| MODULARITY | ---- | PRACTICES INSURING HIERARCHICAL FRAMEWORK OF HIGHLY INDEPENDENT MODULES |
| CONSISTENCY | ---- | PRACTICES WHICH PROMOTE UNIFORMITY OF DESIGN, PROGRAMMING AND DOCUMENTATION |
| SIMPLICITY | ---- | PRACTICES WHICH PROMOTE UNDERSTANDABILITY, AVOIDANCE OF COMPLEXITY AND PROGRAMMING "TRICKS" |
| EXECUTION EFFICIENCY | ---- | PRACTICES WHICH ATTEMPT TO MINIMIZE CPU USE, DISK ACCESSES, ETC. |
| STORAGE EFFICIENCY | ---- | PRACTICES WHICH ATTEMPT TO MINIMIZE STORAGE REQUIREMENTS |
| SOFTWARE SYSTEM INDEPENDENCE | ---- | PRACTICES WHICH MINIMIZE DEPENDENCE ON OPERATING ENVIRONMENT |
| MACHINE INDEPENDENCE | ---- | PROVISIONS FOR ELIMINATING SOFTWARE DEPENDENCE ON HARDWARE |
| EXPANDABILITY | ---- | PRACTICES WHICH READILY ENABLE EXPANSION OF DATA RESOURCES |
| INSTRUMENTATION | ---- | PRACTICES WHICH PROVIDE MEASURMENT OF USAGE OR ERROR IDENTIFICATION |
| ACCURACY | ---- | PROVISIONS FOR PRECISION IN CALCULATIONS AND DISPLAYS |
| TRACEABILITY | ---- | PRACTICES INSURING THREAD FROM REQUIREMENTS TO IMPLEMENTATION |

Figure 2-3. Definitions of Selected Quality Criteria.

| QUALITY CRITERIA | QUALITY FACTORS | | | | | | |
|---|---|---|---|---|---|---|---|
| | MAINT | FLEX | RELIAB | TEST | REUSE | EFFIC | CORREC |
| MODULARITY | X | X | | X | X | | |
| CONSISTENCY | X | X | X | | | | X |
| SIMPLICITY | X | X | X | X | | | |
| EXECUTION EFFICIENCY | | | | | | X | |
| STORAGE EFFICIENCY | | | | | | X | |
| SOFTWARE SYSTEM INDEPENDENCE | | | | | X | | |
| MACHINE INDEPENDENCE | | | | | X | | |
| EXPAND-ABILITY | | X | | | | | |
| INSTRUMENT-ATION | | | | X | | | |
| ACCURACY | | | X | | | | |
| TRACEABILITY | | | | | | | X |

Figure 2-4. Association of Selected Quality Criteria to Quality Factors

## 2.2.3 Quality Standards and Software Metrics

Quality standards then are established for defining the criteria further. A standard may be defined as an agreed upon convention or rule associated with a specific software development activity. For example, formally documented standards addressing the issue of how to comment COBOL source code, when combined, serve as a possible category of programming standards.

Standards and guidelines ultimately can be identified for development activities within all life cycle phases. Standards should be associated with one or more qualiity criteria which in effect become the reasons for the stanaard.

These are examples of standards that can be established for achieving the criterion of simplicity of design and implementation:

1. The total number of procedural statements in each source program will not exceed 75.

2. The number of subroutine calls in any module will not exceed six.

3. No module will be designed to include more than a single iteration (REPEAT-UNTIL, DO-WHILE, PERFORM-UNTIL, etc.). If the need arises for nested iterations, the innermost iteration must be implemented in a subroutine call.

After standards are in place for criteria, software metrics then can be calculated for the purpose of quantifying the degree to which software modules comply with the standards. A possible metric that can be defined for each these three examples simply is the percentage of modules which comply with the standards; the following formula defines this metric:

$$\text{metric} = 1 - \frac{\text{\# modules which violate the rule}}{\text{total \# of modules}}$$

The software metric for simplicity, i.e., an aggregate metric for the entire criteria, can be the average of the metrics for each of these three example standards.

To summarize, each quality factor, e.g., maintainability, is defined in terms of a number of criteria, e.g., simplicity, modularity, consistency, etc.. Each criterion is defined further in terms of standards. Software modules must be checked to determine which modules violate which standards. Metrics then can be calculated for quantifying software quality.

## 2.3 Approach to Software Configuration Management

A software configuration management system provides the mechanisms and controls for managing software. It minimallly must manage the following software mechanisms and controls:

1. Secure storage of tested source code.

2. Change procedures for source code, protecting against concurrent updates and the loss of previous versions.

3. Creation of system releases throughout the development phases and the software maintenance phase.

4. Creation of development and testing environments, including specifications for object and include libraries.

Software configuration management systems may include a problem reporting service, which may be used for assigning problems to developers. This service may be used to document changes in the source code.

A good software configuration management system provides efficient automated procedures for compiling and linking code. These procedures may be used only during periodic releases of the software, or they may be available to developers for their development testing.

If the software is to be used on more than one computer, there must be provisions to handle generic code modules and system dependent modules. These provisions will encompass all aspects of software configuration management, including the storage of the source code and the management of releases.

## 2.4 Approach to Documentation and Life Cycle Activities

A quality software project progresses through various phases and steps, which define the appropriate activities for its life cycle. At each step in the life cycle, there are appropriate documents to be produced. The following items represent the major phases and steps of a project:

1. Understand the problem.
   a. Needs analysis
   b. Requirements definition

2. Formulate and justify the solution.
   a. Preliminary design
   b. Detailed design

3. Construct and integrate the solution.
   a. Construction and verification testing
   b. Integration and validation testing

4. Implement and maintain the solution.
   a. Implement and user acceptance
   b. Maintenance and support

The major objective of phase one is to identify the improvement concepts required for satisfying contractual objectives. High level system requirements for the new software system must be identified. System-level requirements are to include performance objectives, design constraints, etc. In addition, there should be a clear identification of major hardware, software, and organizational elements of the overall system to be developed (i.e., the "to be" system).

Phase two begins with identifying subsystems and their interfaces and ends with completed designs for each subsystem, the following items should be included in the design effort:

a. A structural representation (structure charts) of the framework of software modules required to implement the solution.

b. Procedural descriptions (pseudo code) identifying logic and algorythmic details associated with complex modules identified on the structural charts.

c. Unit/integration test specifications identifying test objectives, required data, test tools, and expected results.

Phase three involves the translation of structure charts, pseudo code, screen formats, and data base design into validated software fulfilling the functional requirements of each of the subsystems. Unit testing and integration testing also is performed during this phase. Integration testing also includes testing procedures for verifying proper integration and coordination of subsystems.

Phase four is a product maintainance phase while the software is being used by customers. This likely will include customizing the software for various users. A special release-specific document must be provided to handle customizing and similar type problems so to prevent overloading user manuals with information oriented to singular user types. It particularly is important during this phase to have an efficient software configuration management system in place to manage the various releases.

## 2.5 Approach to System Testing

### 2.5.1 System Test Plans

Successful unit/integration testing of each individual subsystem does not by itself provide sufficient confidence of overall product integrity. Inevitable refinement and modification of functional requirements and information requirements during the course of the developement process requires that comprehensive system-level testing also be performed.

Satisfactory completion of the implementation phase of all subsystems, therefore, is followed by formal system testing. The primary intent of system testing is to verify that all subsystems function cooperatively as an integrated product. System testing insures proper coordination, information flow, and integration between all major subsystems within the overall system.

Depending on the nature of the development effort, other system-level quality issues also are verified during this phase, including the following type tests:

1. Recovery testing

2. Background testing

3. Stress testing

4. Performance testing

Recovery testing pertains to evaluating system recovery procedures (e.g., rollforward of data base after-images following a hardware failure and restart of transactions that were running at the time of the crash). Where appropriate, system test plans should include adequate provisions for demonstrating recovery capability and recovery procedures.

Background testing involves loading the system with a reasonable (normal) load to observe overall system performance. Load can be generated via a load generator (hardware/software combination) which is external to the system. Background testing provides a mechanism for evaluating two important areas:

a. Observing if system or performance parameters are properly set and

b. Detecting problems unrelated to a single transaction but rather to concurrent execution or interaction among a number of transactions (e.g., reentrancy problems, problems with concurrent access to the data base, etc.).

Stress testing provides an abnormally high load which causes system resources (buffers, concurrent tasks, etc.) to become overloaded. Stress testing is a strategy for testing the upper limits of resource capability.

Performance testing normally is carried out to demonstrate that performance constraints as specified in preliminary and detailed design documentation are being satisfied. Performance objectives should be documented clearly and developers should have a realistic plan for demonstrating and measuring performance characteristics.

The system test plan should be independently produced. The system test plan covers requirements explicitly traceable to functional and information requirements stated in preliminary design documentation and derived requirements not explicitly specified.

The actual system test is jointly carried out by developers and independent system test personnel.

The development of system test plans normally is initiated during the preliminary design phase. User's manuals and subsystem requirements specifications are prerequisites for development of system test plans.

The following items define the contents of a typical software system test plan:

1. Explanation of test design standards and conventions.

2. Provisions for performance testing, background testing, stress testing, and recovery testing.

3. Descriptions of specific test cases if test demonstrates an exercising of the system's total functionality; otherwise, test cases should be defined within a combined description, such as a regression test script file and its function.

4. Order of test execution, especially if there are test dependencies.

5. Procedures for conducting test types, i.e., installation test, regression testing, etc.

6. Descriptions of test data and data base load programs; the description should include what test type uses them.

7. Hardware requirements.

8. Applicable test tools (hardware and software).

## 2.5.2 System Testing Issues

1. SYSTEM-LEVEL TEST CASES

All test cases identified for system testing should be functional rather than structural. Structural path testing is performed during unit/integration testing. System testing should adopt the user's point of view. Each transaction is thought of as a black box. Each test case emphasizes a variation of external input to the transaction.

2. SYNTAX TESTING

For each external input of each transaction (e.g., a specific input field identified on a screen format for a transaction), a sufficient number of test cases should be identified to assure that the transaction is properly reacting to legal and illegal input syntax. Syntax testing is extremely important. A determination of whether or not sufficient test cases have been identified to satisfy the need for adequate syntax testing depends on the nature of the application.

3. SEMANTIC TESTING

For each external input of each transaction, a sufficient number of functional test cases should be identified to assure that a transaction is properly reacting to the semantic content of external input. Like syntax testing, semantic testing also is important because of its immediate visibility to users.

Like syntax testing, a determination of whether or not sufficient test cases have been identified to satisfy the need for semantic testing also depends on the nature of the application and is negotiable. For numerical input values, however, a sufficient number of test cases should be identified to cover crucial input values such as extremes (min, max, min+1, max+1) and excluded values.

4. TRACEABILITY

System testing verifies the correctness of the system in terms of the functional specification portrayed in subsystem requirements specifications and user manuals. Expected results of test cases as formally documented in the test plan should be traceable to functional documentation.

## SECTION 3

## REFERENCE DOCUMENTS

This section provides brief descriptions of the contents of relevant documents. The DAPro 6203 documents are all part of the Final Technical Report.

### 3.1 Software Guidelines

UM 620324000 - Software Development Guidelines

General programming guidelines. Programming standards for COBOL, FORTRAN, and C, including templates. IISS error handling philosophy.

### 3.2 Software Configuration Management

DS 620324000 - SCM Development Specification

The functional requirements of Software Comnfiguration Management. The user and administrative modules and their relationships and purposes. The database files used and their purposes.

CMU620324100 - SCM User's Manual

Overview of the user functions for accessing source code that is in SCM, for entering source code into SCM, and for modifying source code in SCM. Specific instructions for each user function. IISS subdirectories and common object libraries. Supplements providing lists of IISS source code with all SCM attributes.

CMA620324000 - SCM Administrator's Manual

SCM directories with their purposes. Updating the SCM database. Specific VAX release procedures. Specific IBM release procedures.

OM 620324001 - VAX Installation Guide

Steps for installing or building IISS on a VAX from a VAX IISS release magnetic tape.

OM 620324002 - IBM Installation Guide

Steps for installing or building IISS on an IBM from an IBM IISS release magnetic tape.

## 3.3 Documentation Guidelines

IDS150120000C - ICAM Documentation Standards

ICAM life cycle. Documentation standards and formats.

UM 620325000 - DM User's Manual

Functions and structure of documentation management. Steps for using the documentation management functions.

SUM620325000 - FAD Administrator's Guide

Explanation of how the Fully Automated Documentation system works to electronically produce most of the text for a product specification document. The steps for using FAD.

## 3.4 Administrative Guidelines

SUM6203200000 - System Administrator's Guide

Instructions for adding new users, assigning privileges and quotas, and doing backups.

## 3.5 Testing Guidelines

STP620340000 - System Test Plan

Defines the release software and documentation required from development for integration and system-level testing; defines the criteria to be used to qualify the software system for release; defines the test types to be performed and the methodology to be used; defines the test environment, and provides the integration test schedules to be followed from code cutoff to product release.

STR620340000 - System Test Report

Actual detailed results of the complete integration and testing of a software product release. Included are a summary and an analysis of the integration and test activities, a summary of errors reported, resolved, and remaining outstanding, and a list of cautions and recommendations.

## 3.6 Life Cycle Activities

IDS15012000C - ICAM Documentation Standards

ICAM life cycle. Documentation standards and formats.